Background

Programmable logic controllers (PLCs) are embedded computers used to control industrial machinery. A PLC consists of a CPU module, a backplane and assorted input/output (IO) modules. The IO modules convert small low voltage signals to higher voltage signals. These signals can be used to turn on equipment such as lights, heaters or motors.

The GE series one PLC is an older model of PLC developed during the 1980s. The CPU module is based on the Intel 8051 microprocessor. The Intel 8051 microprocessor is an 8 bit microprocessor. The 8051 processor used in the GE CPU module was equipped with 4kB static ram to hold programs. The ram was backed up by a battery to retain programs when power was turned off. The use of battery backed ram was necessary because Flash and EEPROM had not yet become available when the system was designed. The system clock of the GE CPU is 2MHz. This is very slow by today's standards.

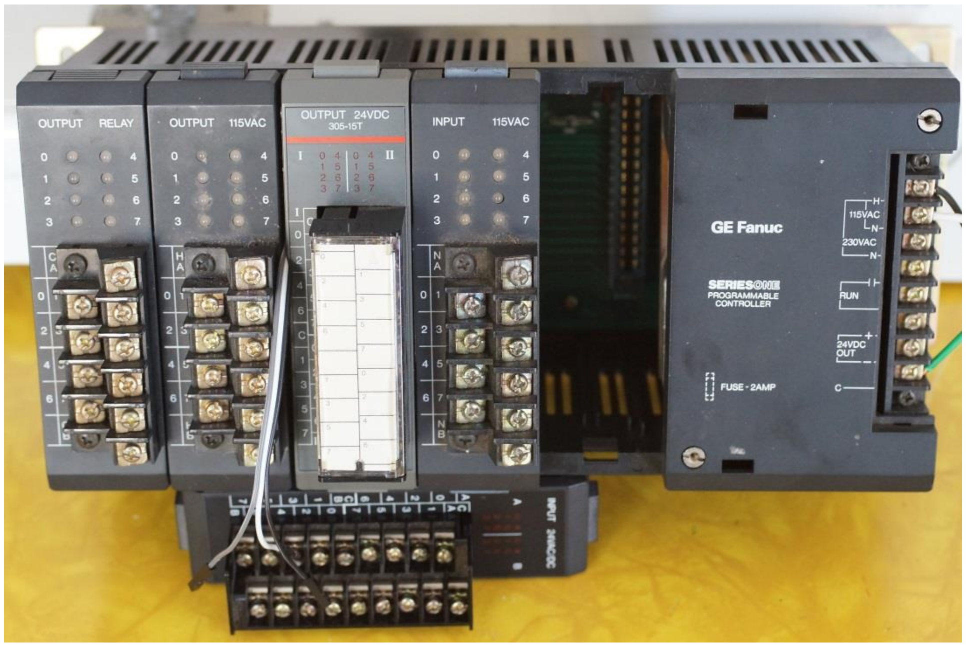

Figure 1 - GE Series One PLC

Programming of the GE CPU was accomplished using a key pad in a process similar to assembly language programming. This process was cumbersome and slow. Programs could only be saved by writing them down on paper or optionally recording them on a cassette tape. Programming from a PC was not possible as the GE CPU lacked a communications port.

The purpose of this project was to design and build a new CPU module for the Ge PLC. The microprocessor chosen for this project is the Texas Instruments Stellaris microcontroller. The Stellaris microcontroller is a 32bit ARM Cortex M4 microprocessor. The highlights of the Stellaris CPU are:

- 80MHz system clock
- 256kB Flash memory for program storage
- 16kB Static ram for program use
- 8 UART ports
- 4 SPI/SSI ports
- 6 64bit timers
- Real time lock

Instead of starting with a bare IC, the Texas Instruments Stellaris Launchpad was used. The Launchpad is a small development board featuring the Stellaris microcontroller. The board includes all the necessary support hardware for the microcontroller, including crystal oscillator, power supply and debug/programming hardware. Programming of the processor is done by connecting the onboard USB micro connector to a PC. This eliminates the need for an expensive external programming device. Interfacing with the launch pad and external circuits is accomplished through two 20 pin headers.

Reverse Engineering

The first part of the development process was to determine the pin out of the back plane. The back plane connects the CPU module with the IO modules. The backplane consists of a circuit board with 6 card edge connectors attached. The first slot is reserved for the CPU module, the rest are available for the IO modules. The connector for the CPU module has pins on both sides of the connector as where the IO module connectors are single sided. Initial attempts to visually trace the circuit traces were unsuccessful because traces couldn’t be followed when being obstructed by components mounted on the board. It was possible to establish which pins on each connector are the grounds for the supply voltages. Figure 2 shows the circuit board of the backplane.

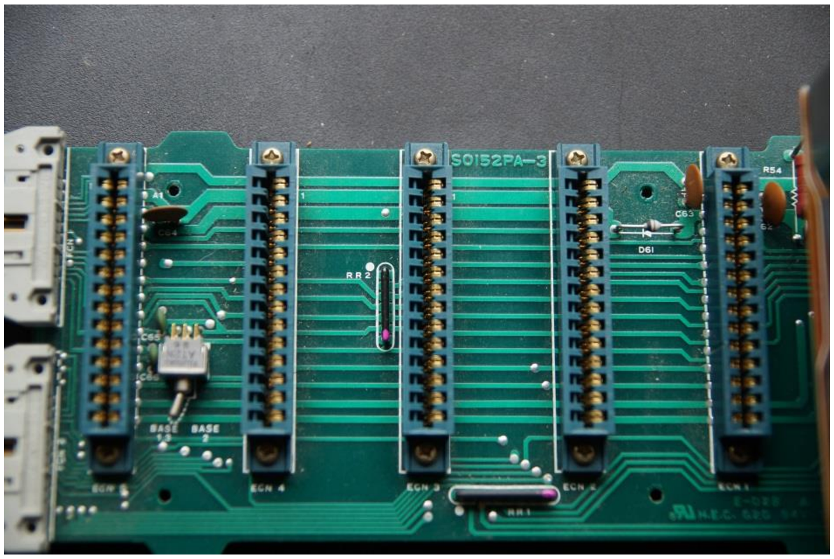

Figure 2 - PLC Back Plane

The next step was to trace continuity between the connectors to determine where each pin connected. This was generally more successful except for 3 pins which were connected in common by a resistor array. The resistor RR2 can be seen in figure 2. The resistor array is a pull up for each line to the +9V bus. Using and ohm meter is was not possible to differentiate between these 3 pins and the 9V common.

The last tracing step was to apply power to the back plane. This revealed which voltages were present on the back plane. It was found the backplane had a 9V bus, 5V bus and a 24V bus. The three pins connected to RR2 were pulled up to the +9V bus. To determine which pin was connected where, each was shorted to ground and the effect was monitored at each connector to determine the destination.

Once the routing of the pins was determined, it was necessary to determine the function of each pin. Determining the function of each pin was relatively simple. The datasheets for the components used on the CPU module and the IO modules were used to determine what signal was likely present on each pin. The address pins were the easiest to reverse trace because each module had an address decoder. The pins to the address decoder were identically connected on each card. The data pin was also the same on each module and relatively easy to identify. The function of each pin is summarized in table 1.

Pin	Slot		
	CPU		IO
1	5V/9Vgnd	5V/9Vgnd	5V/9Vgnd
2	+9V	+9V	+9v
3	+5V	RUN	RUN
4	+5V	Data	Data
5	cs0	WRc	CS
6	cs1	A0	A0
7	cs2	A1	A1
8	cs3	A2	A2
9	cs4	WRA	WRA
10	cs5	WRB	WRB
11	cs6	WRC	WRC
12	cs7	WRD	WRD
13	+24V	+24V	+24V
14	24Vgnd	24Vgnd	24Vgnd

Table 1 – Connector Pin Outs

An interesting finding is that each IO point is addressed individually. There is only one data bit on the back plane. Each bit on each module is individually addressed, instead of addressing a byte at time. This was a surprise since the processor uses an 8bit data memory bus and could address each module in a single write instead of 8.

Voltage Translation

The GE PLC uses 9V on the back plane to communicate between the CPU module and the IO modules. It was expected this would be a 5V bus since the CPU module including the processor and ram operate at 5V. The reason for using 9V is likely noise immunity since PLCs operate in industrial environments where electrical noise is common. The new Stellaris CPU operates a 3.3V and can accept inputs up to 5V. Applying 9V to the pins, even in open collector mode would cause immediate damage to the processor. Converting output voltages from 3.3V to 9V was simple because the original equipment is still available. The original CPU module used a CD4504B IC to convert from low voltage to high voltage. This IC takes a 9V power supply and a 3.3V power supply, inputs at 3.3V are converted to 9V. The CSx bits in table 1 were converted though an LS145 IC. The LS145 is a binary coded decimal decoder. The input is 4 address bits at 3.3V, the outputs are 9 open drain outputs capable of pulling down voltages up to 15V.

While the outputs from the CPU were easy to convert, the data bit to read and write the IO points needed to convert voltages in both directions. A very simple solution was found which only required 1 mosfet. Figure 3 shows the circuit used to convert bidirectional data. To understand the operation you have to look at 4 operating states:

1. When 3.3V Signal is pulled high, VDS = 0V so the mosfet is off, this lets the output go high.
2. When 3.3V Signal is pulled low, VDS = 3.3V so the mostfet is on, this pulls down the output.
3. When 5V Signal is pulled high and 3.3V Signal is in the input state, VDS = 0V so the mosfet is off and the input is high.
4. When 5V Signal is pulled low, the mosfet acts as a diode so 3.3V Signal becomes ~0.7V.

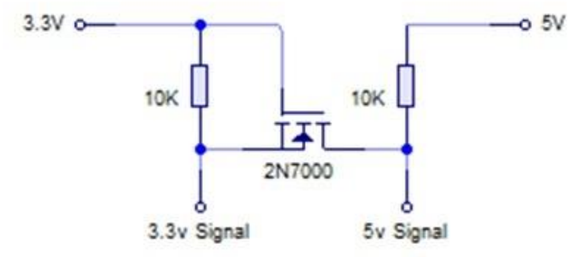

Figure 3 - Bi-Directional Level Converter

Initial Testing

After determining the pin out of the back plane, initial testing was done to verify each input and to determine the sequence for reading/writing to the IO modules. To allow easy access to the card edge connectors, adaptors were made with a card edge on one side and a 0.1" pin header on the other. This allowed test leads to be easily connected. Figure 4 shows the pin out adapters. These were made by applying clear tape to a blank circuit board. The traces for the pins were drawn on the tape with a marker. An Xacto knife was used to remove the tape in areas where there was no marker. This exposed those areas so they could be etched away. The etching was done by mixing 1 part vinegar with 1 part hydrogen peroxide and adding table salt as necessary. This etching mixture works more slowly than commercial etching solutions but is safer to work with. The boards took approximately 4 hours to finish etching. After the etching was completed, the remaining tape was removed and the 0.1" pin headers were soldered in place.

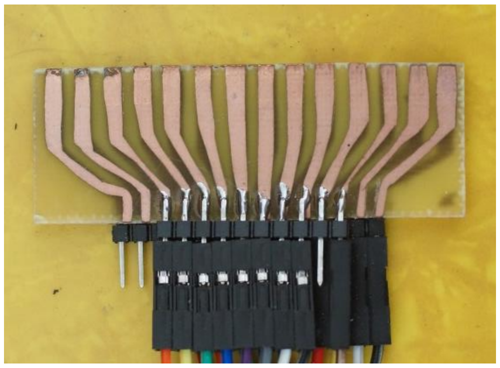

Figure 4 - Connector Adapter

The first testing was done by placing the adapters into each connector on the back plane. A volt meter was connected to each pin on the IO module connector. At the CPU module connector each pin was connected to +9V and GND alternately to verify the pin outs. After all pin outs were confirmed, a prototype board was built to test controlling the IO modules. Figure 5 shows the prototype board. A perforated prototyping board was used to hold all the components to interface from the launch pad to the IO modules. At the center of the board are the 20 pin headers to attach the launch pad. A 3.3V linear voltage regulator was added to supply +3.3V for the launch pad and the logic ICs. The LS145 and CD4504 ICs were soldered in place and connected to pins on the launch pad using 28 gauge wires from a ribbon cable.

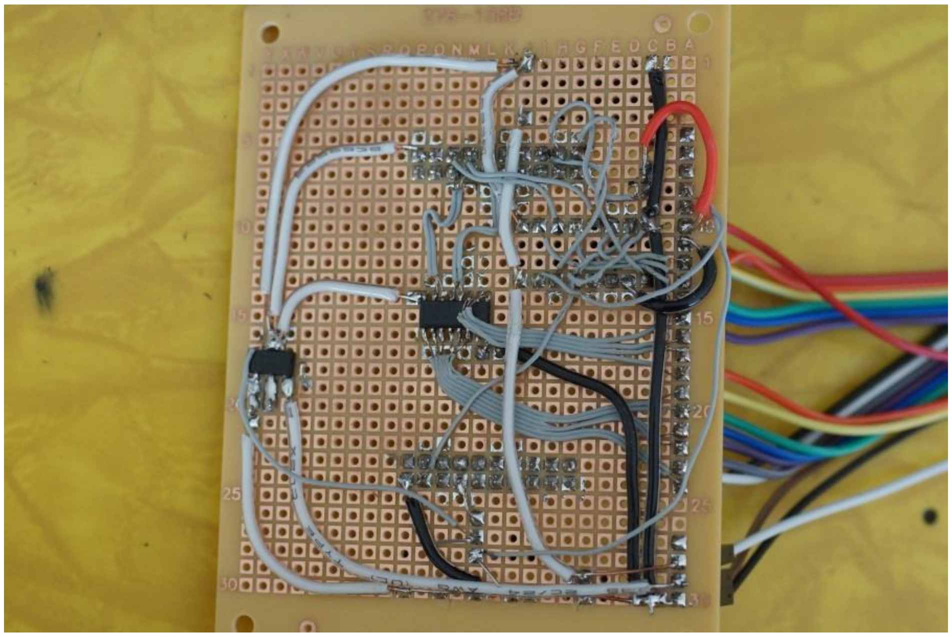

Figure 5 - Prototype Board

A test program was written to set the pins on the launch pad to control the IO modules. An IO module was installed to test setting the outputs on. Initially no response was noted except when the processor was reset and all pins were low. This lead to the realization that the pins were active low. Once the program was setup to active low, the outputs on the IO module immediately turned on. The outputs however did not stay on after the CSx was taken high. A volt meter was connected to IO module pins to verify the signals to the IO module. It was found that the voltage on the RUN pin of the module was not pulling down properly. The voltage at the RUN pin was ~6V at the low logic state and 9V at the high logic state. Manually pulling down the RUN pin to GND set the low logic level to .3V. With this level the outputs on the IO module remained on when the CSx pin was pulled high. An ampere meter was connected between the RUN pin and ground. A current of ~125mA was measured on this pin. This current exceeded output current rating of the CD4504. The original user's manual for the GE PLC shows that the RUN pin on the connector is also connected to a relay in the back plane that turns on when the RUN pin is pulled low. Figure 6 shows the diagram from the user's manual. The diagram shows an open collector transistor being used. The connection from the CD4504 was replaced with an open drain connected 2N7000 mosfet. When the mosfet was turned on, the voltage at the RUN pin was pulled down to .3V. This solved the problem with the IO module outputs turning off when the CSx pin was pulled high.

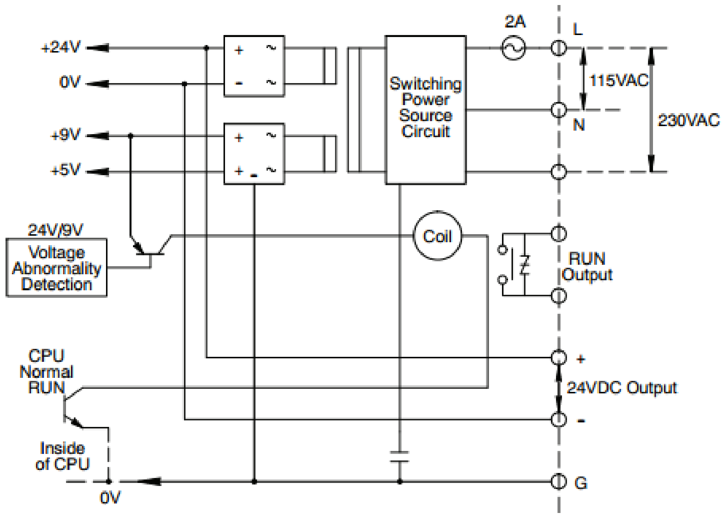

Figure 6 - User's Manual Diagram

The IO module used for the first testing was an 8 channel module. When a 16 channel module was installed, it was observed that both outputs 1 and 9 turned on at the same time, like wise outputs 2 and 10 turned on at the same time and so on. It was discovered that the WRb pin was mistakenly left pulled low. This revealed that to write outputs 1-8 the WRa pin needed to be pulled low and WRb pulled high, to write outputs 9-16, WRb needed to be pulled low while WRa was pulled high. At this point all steps in writing the IO modules were worked out so the process of designing the finished circuit board began.

Circuit Board Design

Due to the poor results when etching the pin adapter boards, it was decided to have the finished circuit boards professionally manufactured. The company Pad2pad was chosen based on lowest cost and the ability to have a custom board shape since the final circuit board would not have a square outline. Pad2pad provides custom circuit board design software for entering the circuit board design. Their software was generally very easy to use and self-explanatory. Using predefined components, you simply select the case type of each component and name each pin of the device with a signal name. When all the parts were placed on the board, the auto routing feature connected all the pins to their destination. There were a few connections left unconnected by the auto routing feature. These were easy to manually route using the wiring tool. Figure 7 shows the resulting circuit board layout with all traces and layers completed. After completing the circuit board design, the layout was electronically submitted to the manufacture to building. The turnaround time for the boards was about 2 weeks. The finished bare circuit board is show in figure 8.

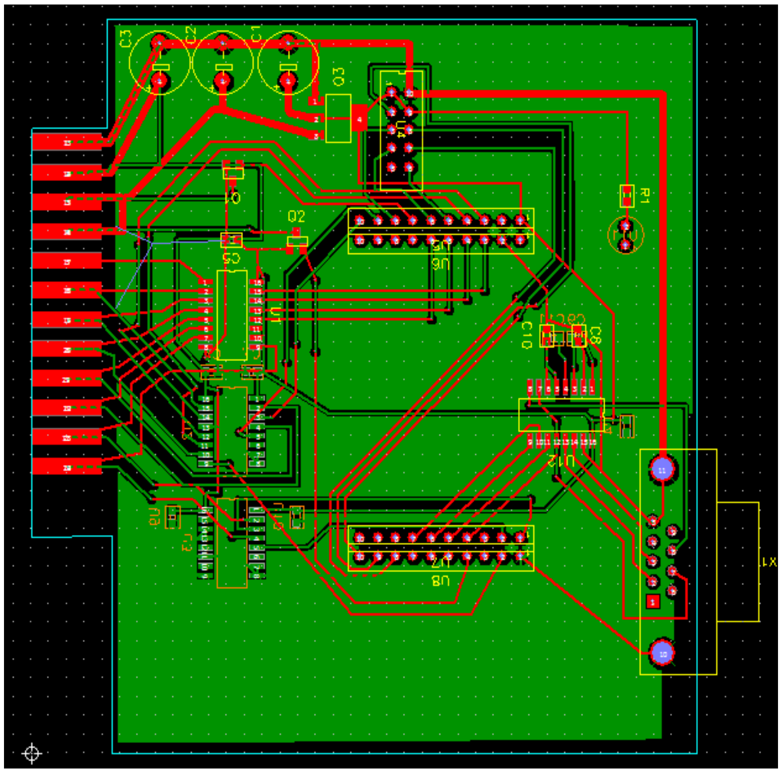

Figure 7 - Circuit Board Layout

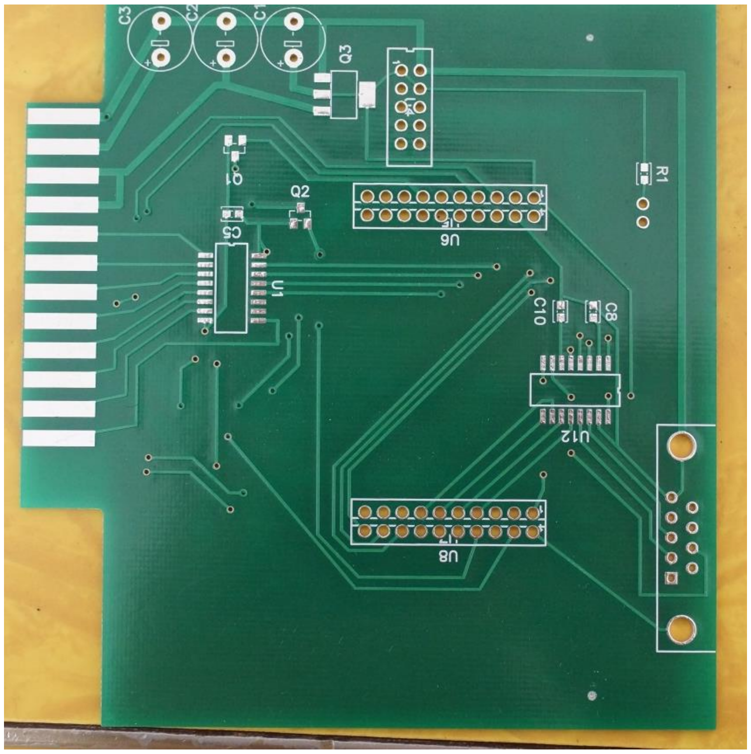

Figure 8 - Finished Circuit Board

Assembly and Testing

The circuit board was soldered and assembled by hand using new components purchased from Mouser.com. The totally soldering time was about an hour. Reflow soldering paste was used to solder the surface mount capacitors because using wire solder often causes the capacitors to stand on end. The ICs were soldered by first soldering opposing corners then soldering each pin individually.

After assembly was completed the launch pad was installed on the pin headers. Programming began with the code written during the initial testing phase. The pins used in the final design were partially different from those used in the initial testing. After reassigning the pins, the same code was able to control the IO modules successfully. Reading inputs in the IO modules however only worked when stepping through the code in the debugger manually. Connecting the DATA pin to an oscilloscope revealed the DATA pin had a settling time of about 75 nano seconds. Figure 9 shows the settling response of the DATA pin. To resolve this, a 30 clock cycle delay was inserted into the code after the IO module output was enabled to allow the DATA pin to settle before reading the input.

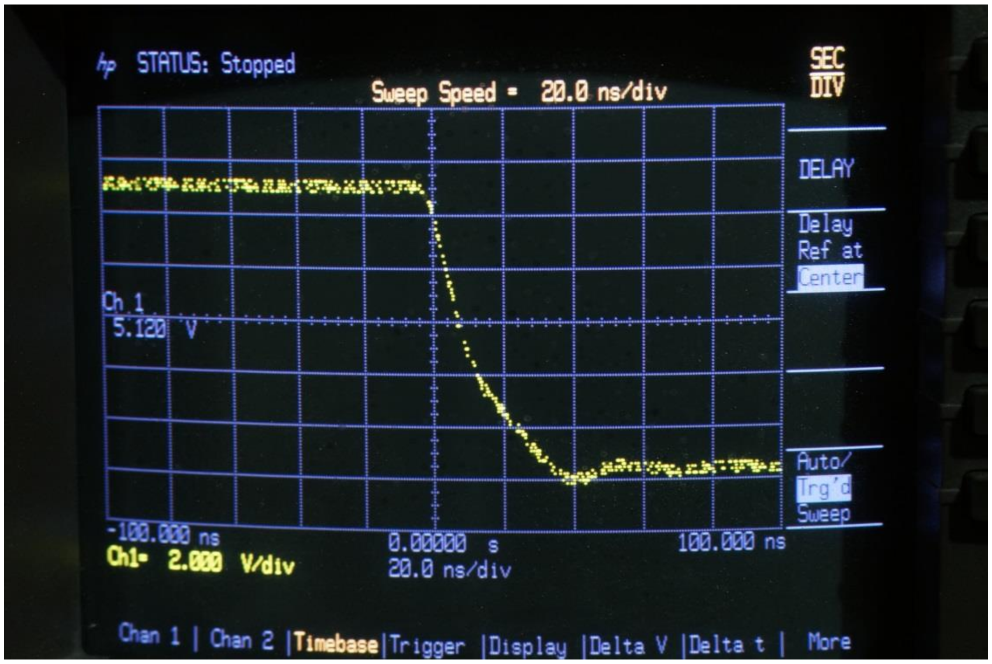

Figure 9 - DATA Output Response

Timers

To regain the full functionality of the original PLC CPU, it required a working timer function block. The Stellaris processor has 6x64bit timers. These are connected to the oscillator of the processor. The processor was configured to run at 66MHz. The timer was configured to roll over when it reached the maximum value a 64bit register can hold. Counting at 66MHz, the timer will roll over at 8862 years. This allows for a high timing precision as well as very long timing intervals. Only 1 timer was used to handle all timers. This allowed for infinitely many individual timers in the user program rather than just 6. To accomplish this, a struct was created which holds the timer length in milliseconds, the status of the timer and the turn on time. The turn on time is a 64bit value which is calculated on the first rising edge enabling the timer. In each scan of the main program loop, the timer current value is compared to the timer turn on time. When the current count exceeds the timer turn on time, the DN bit of the timer status is set, indicating the timer is on. See appendix A for details of the timer code.

Other Features

Not all of the new features of the new CPU module have been implemented or tested in the software. These features are the dual UART RS232 ports and the Ethernet port. The software work to implement these features is more work than the time allotted for this project so far. The system already features one UART through the USB debug connection. This port is enabled by default and appears as a COM port on the programming system. The new COM ports were intended to interface with peripherals in a running machine such as motor drives or bar code scanners. For this reason an RS232 host port type connector was used. Additionally, the Launchpad has a built in USB host port which can host other modern USB devices such as flash drives or even USB COM ports.

Conclusion

An update CPU module for the GE Series One PLC was successfully designed and built. This CPU module features many new modern features. The development of the hardware involved many new learning experiences such as negative logic, finding and reading data sheets, circuit board design integrating hardware peripherals in software and debugging a hardware to software interface. Pictures of the finished CPU module are shown in figures 10 -12.

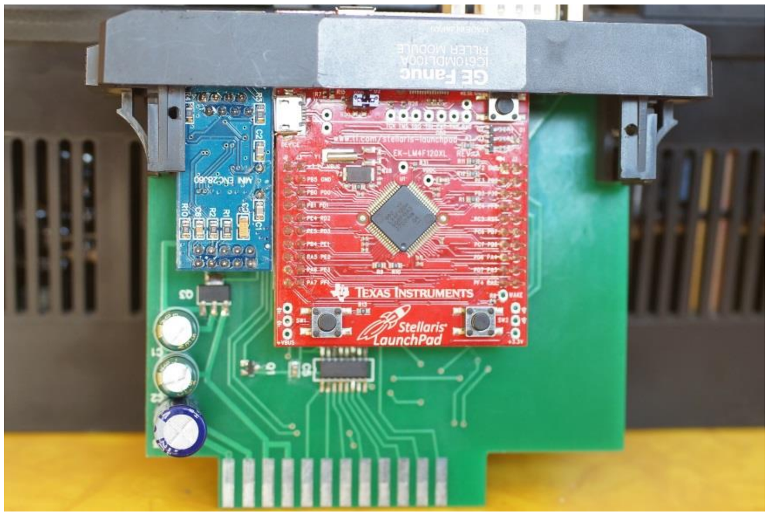

Figure 10 - Finished CPU Module

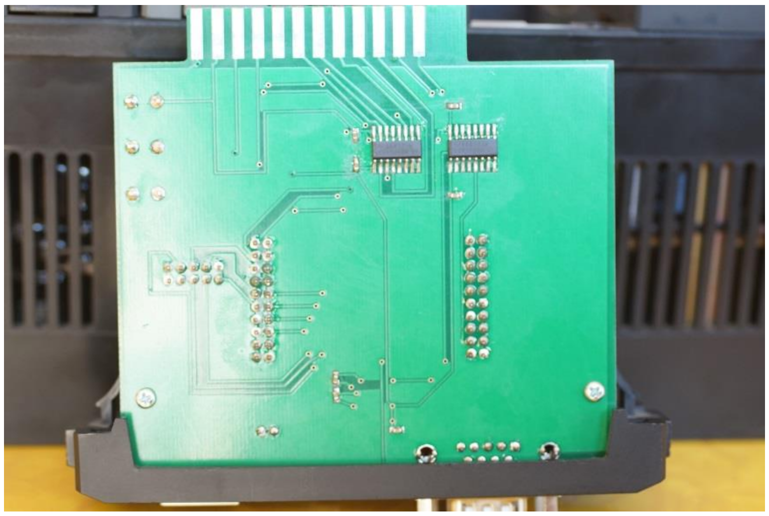
Figure 11 - Bottom of Finished CPU Module

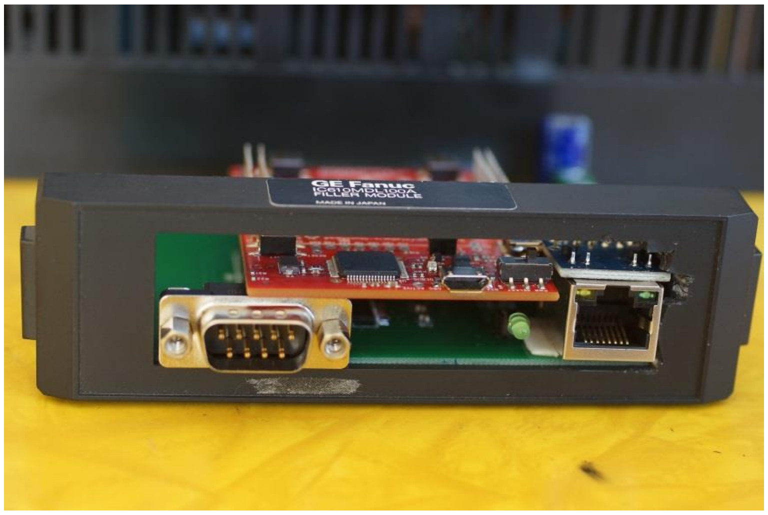

Figure 12 - Front of Finished CPU Module

Appendix A – Source Code

```
#define SET_RUN_ON GPIOPinWrite(GPIO_PORTB_BASE, 0x10, 0x10);
#define SET_RUN_OFF GPIOPinWrite(GPIO_PORTB_BASE, 0x10, 0);
#define SET_INPUT {GPIO_PORTF_DIR_R = 0x00;}
#define SET_OUTPUT {GPIO_PORTF_DIR_R = 0x01;}

//Stucture to be used with the timer fcn.
struct Timer{
  unsigned char en;
  unsigned char dn;
  unsigned long setTime;
  unsigned long long endTime;
}timers[8];

//Completes a time delay
//SetTime is only loaded on the transition of enable from 0 to 1
//Updates after enable goes high will be inored until the next transition
void timer(struct Timer * cTimer,char enable, unsigned long setTime);

//Sets up timer hardware
void timerSetup();

void setupIO();

//Sets the bit number on the selected card
//Should not be used directly
void setBitN(unsigned char bitN);

//Read a single input bit
//Input the bit in hex format
//0xXY where X is the slot and Y is the bit
unsigned char readBit(unsigned char bit);

//Write a single output bit
//Input the bit in hex format
//0xXY where X is the slot and Y is the bit
void writeBit(unsigned char bit,unsigned char data);

/*
 * main.c
 */
int main(void) {
    setupIO();
    timerSetup();
    unsigned char toggle = 0;
    unsigned char input1,input2;

    while(1){
                SET_RUN_ON;

                input1 = readBit(0x00);
                input2 = readBit(0x07);

                writeBit(0x20,readBit(0x00));

                writeBit(0x30,1);
                timer(&timers[0],!timers[0].dn,5000);
		if(timers[0].dn) toggle ^= 1;

		writeBit(0x021,toggle);
		timer(&timers[1],toggle,3000);
		writeBit(0x32,timers[1].dn);
	}

		return 0;
}

void timerSetup(){
		//Set speed to 66MHZ
		 SysCtlClockSet(SYSCTL_SYSDIV_3|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
		 SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER0);
		 SysCtlDelay(10);
		 TimerConfigure(WTIMER0_BASE,TIMER_CFG_ONE_SHOT_UP);
		 TimerDisable(WTIMER0_BASE,TIMER_BOTH);
		 //Set the timer to the max value
		 TimerLoadSet64(WTIMER0_BASE, ~0);//Timer isn't 64 bit but this sets both timers at once.
		 TimerEnable(WTIMER0_BASE, TIMER_A);
}

void setupIO()
{
		volatile unsigned long ulLoop;
		//Enable io ports
		SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF + SYSCTL_RCGC2_GPIOD + SYSCTL_RCGC2_GPIOB
					+ SYSCTL_RCGC2_GPIOA;

		//This just uses a bit of time
		ulLoop = SYSCTL_RCGC2_R;

	//Setup io ports
		//Data IO
	GPIO_PORTF_DIR_R = 0x01;//Set as output
		GPIO_PORTF_LOCK_R = 0x4C4F434B;//Unlock the port conf
	GPIO_PORTF_CR_R = 0x00000001;
	GPIO_PORTF_DEN_R = 0x0d;
	GPIO_PORTF_PUR_R = 0x01;
	GPIO_PORTF_DR8R_R = 0x01;
	GPIO_PORTF_LOCK_R = 0;
	GPIO_PORTF_DATA_R = 0x0c;

	//Card select bits
	GPIO_PORTD_DATA_R = 0;
	GPIO_PORTD_LOCK_R = 0x4C4F434B;//Unlock the port conf
	GPIO_PORTD_CR_R =  0x0f;
	GPIO_PORTD_DIR_R = 0x0f;
	GPIO_PORTD_DEN_R = 0x0f;
	GPIO_PORTD_PUR_R = 0x0f;
	GPIO_PORTD_LOCK_R = 0;


	//Bit select and Run bit
	GPIO_PORTB_DATA_R = 0;
	GPIO_PORTB_DIR_R = 0x17;
	GPIO_PORTB_DEN_R = 0x17;
```

```
    GPIO_PORTB_PUR_R = 0x17;

                //Chip select b
    GPIO_PORTA_DIR_R = 0xe0;
    GPIO_PORTA_DEN_R = 0xe0;
    GPIO_PORTA_PUR_R = 0xe0;
    GPIO_PORTA_DATA_R = 0;

    GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0xC0);
}

void setBitN(unsigned char bitN)
{
                GPIOPinWrite(GPIO_PORTB_BASE, 0x07, ~bitN);
}

unsigned char readBit(unsigned char bit)
{
                unsigned char data;
                unsigned char slot = bit >> 4;
                GPIOPinWrite(GPIO_PORTF_BASE, 0x1, 0);

                SET_INPUT;
                GPIO_PORTD_DATA_R = slot;
                setBitN(bit);

                //Decide which slots to read
                //This may need to be expanded to include WRc and WRd
    if(slot < 8){
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0x80);//Active LOW
    }else{
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0x40);//Active LOW
    }


                SysCtlDelay(10);//Delay 30 cycles to all input to settle in 125nS
                data = GPIO_PORTF_DATA_R & 1;
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0xC0);
                SET_OUTPUT;

                return !data;
}

void writeBit(unsigned char bit, unsigned char data)
{
                unsigned char slot = bit >> 4;
                SET_OUTPUT;
                GPIO_PORTD_DATA_R = slot;
                setBitN(bit);

                //Set output before bit is selected.
                GPIOPinWrite(GPIO_PORTF_BASE, 0x1, data);

                //Decide which slot to write and enable write
    if(slot < 8){
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0x80);
    }else{
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0x41);
    }

                SysCtlDelay(10);//Delay 30 cycles
                GPIOPinWrite(GPIO_PORTA_BASE, 0xC0, 0xC0);//Disable write
}

void timer(struct Timer *ctimer, char enable,unsigned long setTime)
{
  unsigned long long ctime = TimerValueGet64(WTIMER0_BASE);

  if(enable && !ctimer->en){//First time after enable goes high
                ctimer->setTime = setTime;
                ctimer->endTime = SysCtlClockGet();
                ctimer->endTime *= ctimer->setTime;
                ctimer->endTime /= 1000;
                ctimer->endTime += ctime;

                ctimer->en = 1;//Mark timer as enabled
                ctimer->dn = 0;
  }else if(enable && ctimer->en){//Timer is already running.
                if((ctimer->endTime <= ctime) && !ctimer->dn){
                  ctimer->dn = 1;//set done
                }
  }else{
             ctimer->en = 0;
             ctimer->dn = 0;
  }
}
```